Pulp - Story #7659

As a user, orphan cleanup does not block all other tasks

10/06/2020 10:59 PM - bmbouter

Status:	ASSIGNED	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	daviddavis	**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Sprint/Milestone:			
Platform Release:		**Tags:**	
Groomed:	Yes	**Sprint:**	Sprint 96
Sprint Candidate:	Yes	**Quarter:**	Q2-2021

Description

Background

When orphan cleanup runs it blocks all tasks submitted after it, until all workers become idle and then executes the orphan cleanup on the resource manager itself. Also orphan cleanup itself as an operation takes on the order of minutes, it's a non-trivial amount of time.

Problem

On a large Pulp system with many workers running long operations, a user will experience a very significant stalling of all newly submitted work. This occurs even when the user is interested in removing a single piece of content. For stakeholders concerned with tasking throughput on large installations, e.g. galaxy_ng this is not viable.

~~A simple solution~~

Have the orphan cleanup run asynchronously and without any locks. This will allow any worker to work on it in parallel with other Pulp task types.

Handling failures

It's a race between another task associating an orphan with a repository version and orphan cleanup deleting an orphan.

In the case orphan_cleanup wins, plugin writers will need to be told they can no longer guarantee that just because content was there a moment ago, it is there now. I expect the exception can just bubble up to the user, and they user can restart the job at which point code like sync will get it right the second time.

In the case the other task associates an orphan, making it a non-orphan, after the orphan_cleanup has identified it as an orphan, we need to ensure the db will stop orphan_cleanup from deleting it via on_delete=PROTECT.

Proposal for complex solution

Add a new field timestamp_of_interest on the artifact and content models

- This field should be set at the artifact/content creation time
- This field sould be updated whenever we work with the existing artifact/content
- Add a configurable option preserve_orphaned_content = X seconds/minutes. It can be whether a global setting or a parameter to the orphan clean up call.

It is expected that the artifact will be added to the content (unorphaning the artifact) quicker than X seconds/minutes (TBD) from the timestamp_of_interest. Similarly, content will be added to a repo version (unorphaning the content) also within X seconds/minutes(TBD).

Sync Pipeline

timestamp_of_interest This timestamp will be set in the sync pipeline in the QueryExistingArtifacts and QueryExistingContents

When querying for existing artifacts/content and they are found, set the timestamp. If committing this transaction fails due to orphan cleanup committing a transaction that removed the objects having timestamp_of_interest set, retry the exact same transaction again, the second time removed objects will not be found and the pipeline will proceed normally to re-download/re-create the artifact/content in the further stages.

For newly created artifacts and content we need to set this timestamp as well, this way they will be marked 'as-planned-to-be-used'(aka I plan to add this artifact to the content or I plan to add this content to the repo)

Upload

- if one shot upload - handle in 1 transaction artifact and content creation.
- for non 1 shot upload, set the timestamp during content/artifact creation, or in case artifact/content are existing ones, update the timestamp. It is expected users will make their second call to associate the orphaned artifact within X seconds/minutes (TDB) since timestamp_of_interest.

Modify

- set the timestamp on the content specified in the add_content_hrefs to prevent the case when orphan would clean them up in the middle of the running task.

Orphan cleanup logic (orphan clean up task)

Will be able to run in parallel without locks.

Remove Content that:

- has no membership (does not belong to any of the repo versions)
- timestamp_of_interest is older than X seconds(was marked to be used X seconds ago but still does not belong to any of repo versions)

Remove Artifact that:

- has no membership (does not belong to any of the content)
- timestamp_of_interest is older than X seconds(was marked to be used X second ago but still does not belong to any of the content)

Under consideration

some syncs can take long time to finish, because of that we could look at the most recent and long running task before removing orphans

History

#1 - 10/06/2020 11:03 PM - bmbouter

- Description updated

#2 - 10/06/2020 11:04 PM - bmbouter

- Description updated

#3 - 10/07/2020 09:32 PM - iballou

I can't think of any reason why the proposed solution would cause trouble in Katello. We'll just want to make sure there is an obvious path for keeping Pulp data consistent.

#4 - 10/08/2020 01:50 PM - ipanova@redhat.com

In the case orphan_cleanup wins, plugin writers will need to be told they can no longer guarantee that just because content was there a moment ago, it is there now. I expect the exception can just bubble up to the user, and they user can restart the job at which point code like sync will get it right the second time.

In this case, my preference would be to extend Stages API to handle this exception behind the scene, re-download the content, rather than asking users to re-run the sync.

However this does not solve the upload usecase - How do we solve the case when content was initially brought into pulp not by sync operation but upload? I don't think telling we no longer guarantee that just because content was there a moment ago, it is there now will bring a lot of confidence to our users. Our value proposition is to reliably manage content and this statement contradicts to it and it worries me, especially when there is not provided good recovery path.

#5 - 10/08/2020 10:29 PM - bmbouter

ipanova@redhat.com wrote:

> In this case, my preference would be to extend Stages API to handle this exception behind the scene, re-download the content, rather than asking users to re-run the sync. I agree with you on this in the long term, but the complexity it brings I don't think is worth it right now. Consider these reasons and let me know if you still think it is.
>
> 1. I think this will make this a plugin-api breaking change, requiring any plugin with a custom pipeline (which is several) to take additional action to port onto it. This would be breaking because I believe to send content from later stages back to earlier stages would require the stages to reference each other.
> 2. The level of effort goes up significantly with this requirement when we could deliver this improvement later and keep the changes smaller.
>
> However this does not solve the upload usecase - How do we solve the case when content was initially brought into pulp not by sync operation but upload? I don't think telling we no longer guarantee that just because content was there a moment ago, it is there now will bring a lot of confidence to our users. Our value proposition is to reliably manage content and this statement contradicts to it and it worries me, especially when there is not provided good recovery path.

I acknowledge that this is a step backwards in reliability, but I want to make two points about this.

1. If we leave the situation as-is, any pulp system will block all new tasks for a really long time every time. Really long like hours because it has to wait for every task on every worker to stop. Some sync's themselves can take hours, not to mention the many operations running on all workers. This is a huge cost for users to pay and I don't think we can justify that user experience.
2. The upload error case is a race condition, one that would occur I believe rarely and is easily recoverable.

My perspective is that, when I consider the balance between the user experience pain of deleting content (hours, unscalable, and almost unworkable) against the downsides to users (brief and recoverable) this bring overall huge benefits.

#6 - 10/12/2020 09:00 PM - ipanova@redhat.com

bmbouter wrote:

> ipanova@redhat.com wrote:
>
> > In this case, my preference would be to extend Stages API to handle this exception behind the scene, re-download the content, rather than asking users to re-run the sync. I agree with you on this in the long term, but the complexity it brings I don't think is worth it right now. Consider these reasons and let me know if you still think it is.
> >
> > 1. I think this will make this a plugin-api breaking change, requiring any plugin with a custom pipeline (which is several) to take additional action to port onto it. This would be breaking because I believe to send content from later stages back to earlier stages would require the stages to reference each other.
> > 2. The level of effort goes up significantly with this requirement when we could deliver this improvement later and keep the changes smaller.

I agree that current situation with the orphan clean-up is barely acceptable and we need to do something about it. However regardless how much effort will be invested now or later this change will remain a breaking one.

> > However this does not solve the upload usecase - How do we solve the case when content was initially brought into pulp not by sync operation but upload? I don't think telling we no longer guarantee that just because content was there a moment ago, it is there now will bring a lot of confidence to our users. Our value proposition is to reliably manage content and this statement contradicts to it and it worries me, especially when there is not provided good recovery path.
>
> I acknowledge that this is a step backwards in reliability, but I want to make two points about this.
>
> > 1. If we leave the situation as-is, any pulp system will block all new tasks for a really long time every time. Really long like hours because it has to wait for every task on every worker to stop. Some sync's themselves can take hours, not to mention the many operations running on all workers. This is a huge cost for users to pay and I don't think we can justify that user experience.
> > 2. The upload error case is a race condition, one that would occur I believe rarely and is easily recoverable.
>
> My perspective is that, when I consider the balance between the user experience pain of deleting content (hours, unscalable, and almost unworkable) against the downsides to users (brief and recoverable) this bring overall huge benefits.

It is hard to argue against the above mentioned points, but I still feel uneasy on taking a step back on reliability. I would love to hear what other people think.

#7 - 11/30/2020 02:27 PM - ipanova@redhat.com

- Description updated

#8 - 12/02/2020 02:43 PM - ipanova@redhat.com

- Description updated

#9 - 12/08/2020 01:44 PM - daviddavis

- Groomed changed from No to Yes

- Sprint Candidate changed from No to Yes

#10 - 01/12/2021 03:13 PM - daviddavis

- Quarter set to Q1-2021

#11 - 04/16/2021 03:43 PM - ipanova@redhat.com

- Description updated

#12 - 04/16/2021 03:46 PM - daviddavis

- Quarter changed from Q1-2021 to Q2-2021

#13 - 04/20/2021 03:01 PM - daviddavis

- Description updated

#14 - 04/20/2021 03:02 PM - daviddavis

- Description updated

#15 - 04/20/2021 03:03 PM - daviddavis

- Sprint set to Sprint 95

#16 - 04/26/2021 06:04 PM - daviddavis

- Status changed from NEW to ASSIGNED

- Assignee set to daviddavis

#17 - 04/27/2021 04:35 PM - daviddavis

- Status changed from ASSIGNED to NEW

- Assignee deleted (daviddavis)

#18 - 04/29/2021 04:59 PM - daviddavis

- Status changed from NEW to ASSIGNED

- Assignee set to daviddavis

#19 - 04/30/2021 06:13 PM - rchan

- Sprint changed from Sprint 95 to Sprint 96